NCRC-0027-US (9362)

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: MANAGING ALLOCATION OF TEMPORARY AND PERMANENT FILES IN A DATABASE SYSTEM

INVENTORS: GREGORY H. MILBY, STEVEN C. GROLEMUND, AND SUSAN E. CHOO

Express Mail No.: EL669040439US
Date: DECEMBER 8, 2000

Prepared by: Trop, Pruner & Hu, P.C.
8554 Katy Freeway, Ste. 100, Houston, TX 77024
713/468-8880 [Office], 713/468-8883 [Fax]

MANAGING ALLOCATION OF TEMPORARY AND PERMANENT FILES IN A DATABASE SYSTEM

TECHNICAL FIELD

The invention relates to managing allocation of temporary and permanent files in a database system.

BACKGROUND

Database systems are used to collect and store various types of information that are later extracted for viewing or manipulation. Various different types of database systems exist, including relational database systems, object relational database systems, and others. Relational database systems store data in relational tables. When a query, such as a Structured Query Language (SQL) query is received, a subset of the data is extracted from one or more relational tables.

Many typical database systems divide data into different storage regions, such as system storage, permanent storage, and temporary storage. System storage contains data associated with the operation and configuration of the database system. Permanent storage typically is used to store user data. The storage is "permanent" in the sense that stored data remains in storage until deleted or modified by a request to delete or update the data. Temporary storage is typically used to store intermediate or final results of a query. The data in temporary storage is usually discarded after the query has completed so that the temporary storage can be used by other database operations.

The behavior of the database control logic differs when accessing data in the permanent and temporary storage regions. When accessing the permanent storage region, the database control logic usually acquires transaction locks to prevent one operation from interfering with another operation, logs data changes, and maintains context information to enable recovery of data if a system crash occurs. However, such tasks are usually not performed when accessing data in the temporary storage region.

To allow different treatment of the different storage regions, the database control

logic may have to be separated for the different types of files. This allows the database control logic responsible for the temporary storage region to avoid some of the management overhead associated with the permanent storage region. However, the separation of the database control logic for temporary and permanent files creates other inefficiencies. For example, the storage space allocated for temporary storage may not be available for permanent storage, and vice versa.

SUMMARY

In general, a method for use in a database system having a persistent storage device and a non-persistent memory comprises storing a first file management context in the persistent storage device, and storing a second file management context in the non-persistent memory. Both the first and second file management contexts are updated to allocate a permanent file, and the second file management context is updated without updating the first file management context to allocate a temporary file.

Other or alternative features will become apparent from the following description, from the drawings, and from the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of one embodiment of a database system having plural access modules associated with corresponding storage units.

Fig. 2 illustrates maps corresponding to file management contexts for temporary storage and permanent storage in the database system of Fig. 1.

Fig. 3 is a flow diagram of a process performed in the database system in connection with the file management contexts at system startup.

Fig. 4 illustrates mapping of allocation units and storage identifiers in accordance with one example.

Fig. 5 is a flow diagram of a process to perform allocation of temporary and permanent storage, in accordance with an embodiment.

DETAILED DESCRIPTION

In the following description, numerous details are set forth to provide an understanding of the present invention. However, it will be understood by those skilled in the art that the present invention may be practiced without these details and that numerous variations or modifications from the described embodiments may be possible.

Fig. 1 illustrates one example embodiment of a database system 10 that is coupled to a client system 12 over a network 14. Examples of the network 14 include a local area network (LAN), a wide area network (WAN), or a public network (such as the Internet). A user at the client system 12 can issue requests, such as Structured Query Language (SQL) requests, to the database system 10 to extract data stored in the database system 10. SQL is the language of relational databases, and is defined by the American National Standards Institute (ANSI).

In the illustrated embodiment, the database system 10 includes multiple nodes 16A and 16B (two or more). In an alternative embodiment, the database system 12 can also be a single-node system. The nodes 16A and 16B are coupled by an interconnect network 18, which is in turn connected to an interface system 20. The interface system 20 includes a query coordinator 21 to receive queries from the client 12, to parse the received requests, and to generate access requests to one or more data servers 22A, 22B in corresponding nodes 16A, 16B. Each node 16A, 16B includes respective one or more control units 50A, 50B that are coupled to a respective memory 52A, 52B, which can be main memory, a cache, or some other form of storage.

Each data server 22A, 22B controls access to a respective storage unit 30A, 30B. The data servers 22A, 22B can also be referred to as "access modules" for the storage units 30A, 30B. In one embodiment, the database system 10 is a relational database management system or an object relational database management system. User data 34A, 34B are contained in relational tables in respective storage units 30A, 30B. The storage units 30A, 30B also contain respective system tables 32A, 32B for storing system operational and configuration information. In addition, the storage units 30A, 30B contain temporary files 36A, 36B that are used to store intermediate or final results of database operations.

The user data 34 is stored in a permanent storage region of the storage unit 30, while the temporary files 36 are stored in a temporary storage region of the storage unit 30. The user data 34 are stored in "permanent" files. Such files are considered "permanent" in the sense that they are not lost in the event of system crashes, resets, or power cycles. However, permanent files can be changed or deleted by user requests during normal operation of the database system 10. A "file" refers to any apportionment of data in the database system—for example, a file can refer to a row or column of a relational table or to some other apportionment.

In accordance with some embodiments of the invention, the permanent storage region and temporary storage region are part of the same pool of storage resources in each storage unit. For example, the pool of storage resources includes blocks or pages (generally referred to as "storage elements") in the storage unit 30.

By employing file management contexts in accordance with some embodiments, a block or a group of blocks can be selectively allocated to either permanent or temporary storage. Two groups of file management contexts are maintained: a first group 38 (referred to as "persistent file management context") is stored in the storage unit 30; and a second group 26 (referred to as "non-persistent file management context") is stored in the memory 52 of each node 16. The storage unit 30 in one embodiment is a persistent storage unit. A persistent storage unit maintains stored data even if power is removed from the database system or the database system is reset. However, the second group 26 is stored in a non-persistent storage unit, such as the memory 52 in the node 16. Examples of the memory 52 include dynamic random access memories (DRAMs), static random access memories (SRAMs), and the like.

In one embodiment, the file management context in either persistent or non-persistent storage includes two maps: a first map to allocate storage identifiers for uniquely identifying each file created in the database system, and a second map to allocate allocation units, which are blocks or pages of storage in the storage units 30A, 30B. Each map contains flags that can be set to indicate allocation of a storage identifier or allocation unit. The flags can be released to indicate when a storage identifier or allocation unit has been released.

A file management context control routine or module 23 in each data server 22 is capable of accessing the file management context 26 through an application programming interface (API) 24, which defines a collection of API calls. In one example embodiment, the API 24 defines the following methods that are invocable by the file management context control routine 23: a method SET_STORAGE_ID() to allocate a temporary or permanent storage identifier; a method RELEASE_STORAGE_ID() to release a temporary or permanent storage identifier; a method SET_ALLOCATION_UNIT() to allocate blocks of storage; and a method RELEASE_ALLOCATION_UNIT() to release blocks of storage.

To access the storage units 30A, 30B, the data servers 22A, 22B issue requests to respective file systems 28A, 28B. The file systems 28A, 28B then generate commands to the storage units 30A, 30B to access data in the storage units 30A, 30B.

The database system 10 according to the arrangement of Fig. 1 is provided as an example only. In other embodiments, other arrangements of the database system can be utilized. For example, another database system is the Teradata database system from NCR Corporation. The Teradata database system includes a parsing engine that interprets SQL statements and controls the dispatch of requests to virtual access module processors (VAMPs) in one or more nodes. The VAMPs control extraction or manipulation of data stored in respective storage units.

Fig. 2 illustrates one example of file management contexts stored in the database system 10. The persistent version 38 of the file management context is stored in the persistent storage unit 30, while the non-persistent version 26 of the file management context is stored in non-persistent storage 52.

The persistent file management context 38 includes a STORAGE_ID map 106 (to allocate storage identifiers to permanent files) and an ALLOCATION_UNIT map 108 to allocate blocks or pages to permanent files). Each map 106 or 108 includes an array of flags that can be set to a "1" or "0" state. A "1" or active state indicates that a storage ID or an allocation unit has been allocated to a permanent file.

The non-persistent file management context 26 also includes a STORAGE_ID map 110 and an ALLOCATION_UNIT map 112. Whereas the maps 106 and 108 in persistent storage are used to allocate storage IDs or allocation units to permanent files,

the maps 110 and 112 are used to allocate storage IDs or allocation unit to both permanent and temporary files. The persistent maps 106 and 108 are subsets of respective non-persistent maps 110 and 112. Each of the STORAGE_ID and ALLOCATION_UNIT maps 110 and 112 includes flags that can be set to a "1" or "0" state, with some of the flags corresponding to permanent files and others corresponding to temporary files.

Even though the same pool of storage elements are used to store both temporary and permanent files, the control logic in the data server 22 performs different management functions depending on whether a flag that is set in the STORAGE_ID or ALLOCATION_UNIT maps correspond to a temporary file or permanent file. For a permanent file, the control logic in the data server 22 performs transaction locking, logging of data changes made to the maps, and performs tasks to provide data consistency across restart boundaries. Allocation of a permanent file is indicated when the persistent STORAGE_ID map 106 or ALLOCATION_UNIT map 108 is updated. However, when a temporary file is allocated, and the non-persistent maps 110 and 112 are updated without updating the persistent maps 106 and 108, the control logic in the data server 22 does not perform transaction locking, logging of data changes made to the maps, and performing tasks to provide data consistency across restart boundaries.

Storage capacity is made more efficient by combining the pool of storage elements to store both temporary and permanent files. At the same time, management overhead is reduced by performing certain tasks only if permanent files are allocated.

Fig. 4 illustrates how flags in the STORAGE_ID map 106 or 110 and in the ALLOCATION_UNIT map 108 or 112 correspond to storage blocks 250. A first flag 252 in the ALLOCATION_UNIT map is set to the active "1" state, which allocates a storage block 272 (which is referred to as Allocation Unit 1). Similarly, flags 254 and 256 in the ALLOCATION_UNIT map are set to active "1" states to allocate respective block 274 (referred to as Allocation Unit 2) and block 276 (referred to as Allocation Unit X), respectively. Although the example shows one storage block per allocation unit, multiple storage blocks can be in one allocation unit. If multiple blocks are assigned to one allocation unit, then corresponding multiple flags in the ALLOCATION_UNIT map

are set active. Flags 262, 264, and 266 in the STORAGE_ID map are set to the active "1" state to allocate storage identifiers to Allocation Units 1, 2, and X.

Fig. 3 shows tasks performed by the data server 22 at system startup as they relate to the file management contexts 26 and 38. The file management context control routine 23 in each node 16 that is starting up reads (at 202) a persistent version of the STORAGE_ID map 106 contained in the storage unit 30 and loads (at 204) the map into the memory 52 of the node 16. The map loaded into the memory 52 is the non-persistent STORAGE_ID map 110. The file management context routine 23 also reads (at 206) the persistent version of the ALLOCATION_UNIT map 108 and loads (at 208) the map into the memory 52 of the node 16. This loaded map is the non-persistent ALLOCATION_UNIT map 112.

In response to requests received from the query coordinator 21, the data server 22 issues calls to the file management context control routine 23 to perform storage allocation. Each storage management call contains a flag that identifies the class of storage desired: temporary or permanent. As shown in Fig. 5, when the file management context control routine 23 receives (at 302) a storage management call, it determines (at 304) if the flag in the storage management call indicates a temporary file or a permanent file. If the flag indicates a temporary file, then the file management context routine 23 searches (at 306) the in-memory or non-persistent STORAGE_ID map 110 to find a free storage identifier. The routine 23 then sets (at 308) the corresponding bit in the non-persistent STORAGE_ID map to an active state. This is accomplished by calling the SET_STORAGE_ID() method in the API 24 (Fig. 1). Next, the routine 23 searches (at 310) the non-persistent ALLOCATION_UNIT map 112 to find a free block or a group of free blocks. The corresponding bit(s) in the non-persistent ALLOCATION_UNIT map 112 are then set (at 312) to the active state by calling the SET_ALLOCATION_UNIT() method. Plural bits are set if the allocation unit contains plural storage blocks.

For temporary files, only the non-persistent maps 110 and 112 are updated; the persistent maps 106 and 108 remain unchanged. When a temporary file is deleted, the API methods RELEASE_STORAGE_ID() and RELEASE_ALLOCATION_UNIT() are called to release flags (set the flags to the inactive state) in the maps 110 and 112.

However, if the flag in the storage management call indicates a permanent file (as detected at 304), then the file management context control routine 23 searches (at 320) the non-persistent STORAGE_ID map 110 for a free storage identifier. The corresponding bit in the non-persistent STORAGE_ID map 110 is then set (at 322) to an active state. In addition, the file management context control routine 23 also sets (at 324) a corresponding bit in the persistent STORAGE_ID map 106 to an active state. Setting flags in the STORAGE_ID maps 106 and 108 are accomplished with respective calls of the SET_STORAGE_ID() method.

Next, the routine 23 searches (at 326) the non-persistent ALLOCATION_UNIT map 112 for a free block or a group of free blocks. The routine 23 then sets (at 328) the corresponding bit(s) in the non-persistent ALLOCATION_UNIT map 112 to an active state. The corresponding bit in the persistent ALLOCATION_UNIT map 108 is also set (at 330) in the persistent ALLOCATION_UNIT map to the active state. Setting flags in the ALLOCATION_UNIT maps are accomplished by calling the SET_ALLOCATION_UNIT() method.

In the operation discussed above, when the control logic in each data server 22 detects that the persistent maps 106 and 108 are updated (indicating that a permanent file is being allocated), the control logic performs transaction locking, data change logging, and other tasks to provide data consistency across restart boundaries. Such tasks are not performed when a temporary file is allocated.

The persistent maps 106 and 108 can be considered to be "shadows" of the non-persistent maps 110 and 112. In the event of a system crash or reset, the persistent maps 106 and 108 are maintained in the persistent storage units 30A, 30B so that permanent files are not lost as a result of the system crash or reset. However, the non-persistent maps 110 and 112 are lost, which effectively erases any allocated temporary files.

The various software routines or modules discussed above are executable on corresponding one or more control units in the database system 10. Each of the control units includes a microprocessor, a microcontroller, a processor card (including one or more microprocessors or microcontrollers), or other control or computing devices. As used here, a "controller" refers to hardware, software, or a combination of both. A

"controller" can refer to a single component or to plural components (whether software or hardware).

The storage units or devices referred to herein include one or more machine-readable storage media for storing data and instructions. The storage media include different forms of memory including semiconductor memory devices such as dynamic or static random access memories (DRAMs or SRAMs), erasable and programmable read-only memories (EPROMs), electrically erasable and programmable read-only memories (EEPROMs) and flash memories; magnetic disks such as fixed, floppy and removable disks; other magnetic media including tape; and optical media such as compact disks (CDs) or digital video disks (DVDs). Instructions that make up the various software routines or modules are stored in respective storage units. The instructions when executed by a respective control unit cause the corresponding system to perform programmed acts.

The instructions of the software routines or modules are loaded or transported to each system in one of many different ways. For example, code segments including instructions stored on floppy disks, CD or DVD media, a hard disk, or transported through a network interface card, modem, or other interface device are loaded into the system and executed as corresponding software routines or modules. In the loading or transport process, data signals that are embodied in carrier waves (transmitted over telephone lines, network lines, wireless links, cables, and the like) communicate the code segments, including instructions, to the system. Such carrier waves are in the form of electrical, optical, acoustical, electromagnetic, or other types of signals.

While the invention has been disclosed with respect to a limited number of embodiments, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover such modifications and variations as fall within the true spirit and scope of the invention.